JOINT INVENTORS Docket No. 20002/16812
P16812

"EXPRESS MAIL" mailing label No.
EV 309991867 US
Date of Deposit: **June 27, 2003**

I hereby certify that this paper (or fee) is being deposited with the United States Postal Service "EXPRESS MAIL POST OFFICE TO ADDRESSEE" service under 37 CFR §1.10 on the date indicated above and is addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450

M. Greer
Magda Greer

APPLICATION FOR UNITED STATES LETTERS PATENT

SPECIFICATION

TO ALL WHOM IT MAY CONCERN:

Be it known that We, **Mark Doran**, a citizen of United Kingdom, residing at 4605 33rd Court NE, Olympia, Washington, 98516; and **Michael Rothman**, a citizen of United States, residing at 3311 11th Avenue Court. NW, Gig Harbor, Washington, 98335; and **Hung Tran**, a citizen of United States, residing at 26011 Galway Drive, Lake Forest, California, 98335; and **Vincent Zimmer**, a citizen of United States, residing at 1937 South 369th Street, Federal Way, Washington, 98003; and **Andy Miga**, a citizen of United States, residing at 231 Clay Court, Olympia, Washington, 98513 have invented a new and useful **METHODS AND APPARATUS TO PROTECT A PROTOCOL INTERFACE**, of which the following is a specification.

METHODS AND APPARATUS TO PROTECT A PROTOCOL INTERFACE

TECHNICAL FIELD

[0001] The present disclosure relates generally to firmware, and more particularly, to methods and apparatus to protect a protocol interface.

BACKGROUND

[0002] The initial phases of computer or processor system operation (i.e., prior to the booting of an operating system by the processor system) following a power-up or a reset are controlled by a basic input/output system (BIOS). In general, the BIOS is implemented as software or firmware in the form of machine readable instructions that are stored in a non-volatile memory coupled to a processor. Following a reset operation or the application of power to the processor, the processor executes the BIOS instructions. Typically, the BIOS performs one or more hardware and software configuration and test activities prior to booting the operating system. The configuration activities carried out by the BIOS are responsible for establishing the manner in which hardware devices (e.g., disk drives, video controllers, keyboard, mouse, etc.) associated with the processor system interact with the operating system executed by the processor system. The test activities collect system configuration information that may be later used, for example, by the operating system to determine that hardware or devices associated with the system are ready for use and to facilitate debugging activities, configuration management activities, etc.

[0003] Typically, certain protocol interfaces such as architectural protocols (APs) should only be called by designated program(s). In a driver execution environment (DXE) phase, for example, protocol interfaces should only be called by a DXE core. However, some protocol interfaces may be inadvertently or maliciously accessed by

callers other than the DXE core. That is, a protocol interface may be incorrectly called and/or replaced. For example, a driver may discover a protocol interface and replace the authentic version of the protocol interface that was previously installed with a fake version of the protocol interface. In another example, the driver may turn off services provided by the protocol interface while the DXE core is operating under the assumption that those services are enabled. As a result, the protocol interface may not be available when an authorized call of the protocol interface is placed.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] FIG. 1 is a block diagram representation of an example processor system.

[0005] FIG. 2 is a block diagram representation of an example architectural hierarchy of the example processor system shown in FIG. 1.

[0006] FIG. 3 is a block diagram representation of an example layout of drivers and protocol interfaces.

[0007] FIG. 4 is a flow diagram representation of example machine readable instructions that may protect a protocol interface.

[0008] FIG. 5 is a flow diagram representation of example machine readable instructions that may process a driver request.

[0009] FIG. 6 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

[0010] FIG. 7 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

[0011] FIG. 8 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

[0012] FIG. 9 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

[0013] FIG. 10 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

[0014] FIG. 11 is a flow diagram representation of alternative example machine readable instructions that may process the driver request.

DETAILED DESCRIPTION

[0015] Although the following discloses example systems including, among other components, software or firmware executed on hardware, it should be noted that such systems are merely illustrative and should not be considered as limiting. For example, it is contemplated that any or all of the disclosed hardware, software, and/or firmware components could be embodied exclusively in hardware, exclusively in software, exclusively in firmware or in some combination of hardware, software, and/or firmware.

[0016] FIG. 1 is a block diagram of an example processor system 100 adapted to implement the methods and apparatus disclosed herein. The processor system 100 may be a desktop computer, a laptop computer, a notebook computer, a personal digital assistant (PDA), a server, an Internet appliance or any other type of computing device.

[0017] The processor system 100 illustrated in FIG. 1 includes a chipset 110, which includes a memory controller 112 and an input/output (I/O) controller 114. As is well known, a chipset typically provides memory and I/O management functions, as well as a plurality of general purpose and/or special purpose registers, timers, etc. that are accessible or used by a processor 120. The processor 120 is implemented using one or more in-order processors. For example, the processor 120 may be implemented using one or more of the Intel® Pentium® family of microprocessors, the Intel® Itanium® family of

microprocessors, Intel® Centrino® family of microprocessors, and/or the Intel XScale® family of processors. In the alternative, other processors or families of processors may be used to implement the processor 120.

[0018] As is conventional, the memory controller 112 performs functions that enable the processor 120 to access and communicate with a main memory 130 including a volatile memory 132 and a non-volatile memory 134 via a bus 140. The volatile memory 132 may be implemented by Synchronous Dynamic Random Access Memory (SDRAM), Dynamic Random Access Memory (DRAM), RAMBUS Dynamic Random Access Memory (RDRAM), and/or any other type of random access memory device. The non-volatile memory 134 may be implemented using flash memory, Read Only Memory (ROM), Electrically Erasable Programmable Read Only Memory (EEPROM), and/or any other desired type of memory device.

[0019] The processor system 100 also includes an interface circuit 150 that is coupled to the bus 140. The interface circuit 150 may be implemented using any type of well known interface standard such as an Ethernet interface, a universal serial bus (USB), a third generation input/output interface (3GIO) interface, and/or any other suitable type of interface.

[0020] One or more input devices 160 are connected to the interface circuit 150. The input device(s) 160 permit a user to enter data and commands into the processor 120. For example, the input device(s) 160 may be implemented by a keyboard, a mouse, a touch-sensitive display, a track pad, a track ball, an isopoint, and/or a voice recognition system.

[0021] One or more output devices 170 are also connected to the interface circuit 150. For example, the output device(s) 170 may be implemented by display devices (e.g., a light emitting display (LED), a liquid crystal display (LCD), a cathode ray tube (CRT)

display, a printer and/or speakers). The interface circuit 150, thus, typically includes, among other things, a graphics driver card.

[0022] The processor system 100 also includes one or more mass storage devices 180 configured to store software and data. Examples of such mass storage device(s) 180 include floppy disks and drives, hard disk drives, compact disks and drives, and digital versatile disks (DVD) and drives.

[0023] The interface circuit 150 also includes a communication device such as a modem or a network interface card to facilitate exchange of data with external computers via a network. The communication link between the processor system 100 and the network may be any type of network connection such as an Ethernet connection, a digital subscriber line (DSL), a telephone line, a cellular telephone system, a coaxial cable, etc.

[0024] Access to the input device(s) 160, the output device(s) 170, the mass storage device(s) 180 and/or the network is typically controlled by the I/O controller 114 in a conventional manner. In particular, the I/O controller 114 performs functions that enable the processor 120 to communicate with the input device(s) 160, the output device(s) 170, the mass storage device(s) 180 and/or the network via the bus 140 and the interface circuit 150.

[0025] While the components shown in FIG. 1 are depicted as separate functional blocks within the processor system 100, the functions performed by some of these blocks may be integrated within a single semiconductor circuit or may be implemented using two or more separate integrated circuits. For example, although the memory controller 112 and the I/O controller 114 are depicted as separate functional blocks within the chipset 110, persons of ordinary skill in the art will readily appreciate that the memory controller 112 and the I/O controller 114 may be integrated within a single semiconductor circuit.

[0026] In the example of FIG. 2, the illustrated architectural hierarchy 200 of the processor system 100 includes hardware 210, a BIOS 220, an extensible firmware interface (EFI) 230, and an operating system (OS) 240. Persons of ordinary skill in the art will readily recognize that hardware 210 may include any physical aspect of the processor system 100 such as the processor 120 and the main memory 130. Hardware 210 also includes the interface circuit 150, input device(s) 160, output devices 170, and/or the mass storage device 180. Basically, hardware 210 is any or all of the components shown in FIG. 1. The BIOS 220 may be implemented as software, firmware or machine readable instructions configured to boot up (i.e., start up) the processor system 100 in a conventional manner. To boot the OS 240 (e.g., Windows® and/or Linux) and to run pre-boot applications, the BIOS 220 manages data flow between the hardware 210 of the processor system 100 via the EFI 230. Alternatively, the BIOS 220 may directly communicate with the OS 240 without the EFI 230 in a conventional manner.

[0027] After power-up, the processor 120 executes instructions of the BIOS 220 to boot the processor system 100. In particular, the processor 120 may load drivers to install and/or initialize components in the processor system 100 such as disk drives, video controllers, keyboard, mouse, etc. The processor 120 may also call installed protocol interfaces to enable components in the processor system 100.

[0028] In the example of FIG. 3, the processor system 100 includes drivers 310, APs 320, and a phase core 330. The drivers 310 are codes configured to produce protocol interfaces to enable components in the processor system 100. For example, the processor 120 may load drivers 310 such as a driver for a keyboard (i.e., Driver #1 312), a driver for a printer (i.e., Driver #2 314), a driver for a monitor (i.e., Driver #3 316), etc. to enable those devices in the processor system 100.

[0029] The APs 320, generally shown as AP #1 322, AP #2 324, and AP #3 326, provide the capabilities of the processor system 100. For example, APs 320 may include a security AP, a metronome AP, a boot device selection (BDS) AP, a runtime AP, a variable write AP, a monotonic counter AP, a status code AP, a central processing unit (CPU) AP, a time AP, a watchdog AP, a variable AP, a reset AP, and a real time clock AP of the processor system 100. Persons of ordinary skill in the art will readily recognize the services provided by each of the listed APs and that others APs may be included.

[0030] The phase core 330 consists of data tables containing platform-related information to filter communication between the drivers 310 and the APs 320. For example, the phase core 330 may be configured to operate during a pre-EFI initialization (PEI) phase (i.e., before the main memory 130 is enabled) and/or during a driver execution environment (DXE) phase (i.e., after the main memory 130 is enabled) to boot up the processor system 100. The phase core 330 also includes boot and runtime services that are available to the OS 240 of the processor system 100. Accordingly, the phase core 330 provides a standard environment for booting the OS 240 and running pre-boot applications.

[0031] In addition, the phase core 330 may directly call the APs 320 (e.g., an application program interface (API) call) and determine whether the drivers 310 may access one or more APs 320. In particular, the phase core 330 may also process a handle protocol associated with one of the drivers 310. The handle protocol indicates a list of one or more protocols that can respond to requests of a variety of services. The phase core 330 uses a globally unique identifier (GUID) (e.g., a 128-bit value) to differentiate services and structures in the boot services environment. A protocol consists of the 128-bit GUID and a protocol interface structure. The protocol interface structure contains the

functions and data used to access a device. Thus, the phase core 330 may differentiate between the APs 320 based on the GUIDs.

[0032] Without the supervision of the phase core 330, however, one of the drivers 310 may discover and inadvertently or maliciously call 360 one or more APs 320. Thus, a set of services provided by one or more APs 320 may be disabled by of the unauthorized call 360. For example, Driver #2 314 may directly call AP #2 334 (e.g., CPU AP) and disable interrupt services via AP #2 334 when the phase core 330 is operating under the assumption that interrupt services are enabled. Further, the APs 320 may be inadvertently or maliciously replaced by one of the drivers 310. For example, Driver #2 314 may attempt to uninstall and replace AP #2 334. Alternatively, Driver #2 314 may be installed in addition to AP #2 334, and the phase core 330 may incorrectly call Driver #2 314 rather than calling AP #2 334. As a result, the phase core 330 may properly call 350 one or more APs 320 whereas none of the drivers 310 may directly call 360 any of the APs 320.

[0033] To prevent APs 320 from being called inadvertently or maliciously, the processor 120 may designate certain APs 320 as restricted protocol interfaces. For example, AP #1 332 may be designated as a restricted protocol interface by being assigned with a SINGLE_CONSUMER tag so that only one caller (e.g., the phase core 330) may call on AP #1 332. As other examples, AP #1 332 may be assigned with either a NO_INSTALL tag or a NO_PEERS tag. The NO_INSTALL tag prevents a protocol associated with a driver from removing the restricted protocol interface (i.e., uninstall) and substituting another protocol for the restricted protocol interface. Accordingly, reinstall operation requested by the driver fails when executed against the restricted protocol interface. The NO_PEERS tag prevents other protocols with the same GUID as the restricted protocol interface from being installed. That is, a version of the protocol

associated with the driver is already installed (i.e., the restricted protocol interface) so the install operation requested by the driver fails when executed against the restricted interface. As a result, restricted protocol interfaces of the processor system 100 protected from unauthorized access (i.e., access by a caller other than the phase core 330).

[0034] Machine readable instructions that may be executed by the processor system 100 (e.g., via the processor 120) to implement protecting a protocol interface are illustrated in FIG. 4. Persons of ordinary skill in the art will appreciate that the instructions can be implemented in any of many different ways utilizing any of many different programming codes stored on any number of many computer-readable mediums such as a volatile or nonvolatile memory or other mass storage device (e.g., a floppy disk, a CD, and a DVD). For example, the machine readable instructions may be embodied in a machine-readable medium such as a programmable gate array, an application specific integrated circuit (ASIC), an erasable programmable read only memory (EPROM), a read only memory (ROM), a random access memory (RAM), a magnetic media, an optical media, and/or any other suitable type of medium. Further, although a particular order of steps is illustrated in FIG. 4, persons of ordinary skill in the art will appreciate that these steps can be performed in other temporal sequences. Again, the flow chart 400 is merely provided as an example of one way to program the processor system 100 to protect a protocol interface.

[0035] In the example of FIG. 4, the processor system 100 is booted up by initializing the phase core 330 (block 410). As noted above, for example, the phase core 330 may be either a PEI core or a DXE core of firmware in the processor system 100. The processor 120 (e.g., via the phase core 330) then determines whether to load a driver for a component in the processor system 100 (block 420). When a driver that needs to be loaded and executed, the phase core 330 serves as a dispatcher to load and invoke the

driver. Accordingly, the processor 120 loads a driver install interface for the phase core 330 to control the driver (block 430). Through the driver install interface, the phase core 330 governs which protocols associated with drivers are installed.

[0036] Typically, the processor 120 loads drivers one at a time. Accordingly, the processor 120 returns to block 420 to determine whether there are additional drivers to load for execution. If not, the processor 120 proceeds to determine whether a driver request was received (block 440). If there is no driver request to process then the processor 120 continues to boot the OS 210 so that control of the processor system 100 may be transferred over to the OS 210 (block 470).

[0037] On the other hand, if the processor 120 receives a driver request to process then the processor 120 begins to process the driver request as described in detail below (block 450). Not only does the phase core 330 determine which protocols associated with loaded drivers to install, but it also governs whether to permit calls to protocol interfaces that were previously installed. After processing the driver request, the processor 120 determines if there are more driver requests to process (block 460). If so, the processor 120 returns to block 450 to process other driver requests. Otherwise, the processor 120 proceeds to block 470 to continue booting the OS 210.

[0038] To process a driver request of a loaded driver, the processor 120 determines whether the driver request is associated with a violation condition of a protocol interface to prevent the protocol interface from being called on inadvertently or maliciously. In the example of flow chart 500 in FIG. 5, the processor 120 may determine if the driver request is associated with a violation condition by determining whether the driver request is a request to access a restricted protocol interface (e.g., a platform password protocol previously installed to the processor system 100) by the loaded driver (block 510). In particular, the processor 120 may process a handle protocol associated with the loaded

driver. As noted above, the handle protocol associated with the loaded driver points to a list of one or more protocols configured to respond to requests for services. The loaded driver may attempt to access the restricted protocol interface. For example, the restricted protocol interface may be accessible by only one caller such as the phase core 330 to prevent the restricted protocol interface from being corrupted. Accordingly, the processor 120 protects the restricted protocol interface from the loaded driver by storing the restricted protocol interface in a protocol database (i.e., hiding the restricted protocol interface from the loaded driver) (block 515). For example, the protocol database may be a data structure stored in the main memory 130 and/or one or more mass storage devices 180. The protocol database may be configured to store one or more restricted protocol interfaces.

[0039] If the processor 120 determines that the driver request is not a request to access the restricted protocol interface by the loaded driver, then the processor 120 determines whether the driver request is a reinstall request by the loaded driver (block 520). That is, the loaded driver may attempt a call directly to the restricted protocol interface (e.g., one shown as 350 in FIG. 3). As noted above, the phase core 330 should make the call to the restricted protocol interface rather than the loaded driver to protect the restricted protocol interface from being replaced and/or corrupted because the reinstall request allows the loaded driver to uninstall and/or substitute the restricted protocol interface. If the driver request is a reinstall request, then the processor 120 may reject the reinstall request by the loaded driver (block 525).

[0040] On the other if the driver request is not a reinstall request, the processor 120 may determine if the driver request is an install request by the loaded driver (block 530). In particular, the loaded driver may request to add a new protocol to the processor system 100. If the restricted protocol interface is the same as the new protocol associated with

the loaded driver (i.e., the restricted protocol interface has the same GUID as the new protocol), the processor 120 may reject the install request by the loaded driver to avoid duplicates (block 525).

[0041] While FIG. 5 is depicted to include blocks 510, 520, and 530 to determine if the loaded driver is associated with a violation condition, the processor 120 may also determine if the loaded driver is associated with a violation condition by implementing only one of blocks 510, 520, and 530. Referring to flow chart 600 of FIG. 6, for example, the processor 120 may determine whether the driver request is a request by the loaded driver to access a restricted protocol interface installed in the processor system 100 (block 610). If the driver request is a request by the loaded driver to access a restricted protocol interface, the processor 120 may protect the restricted protocol interface from the loaded driver by storing the restricted protocol interface in a protocol database (i.e., hiding the restricted protocol interface from the loaded driver) (block 615).

[0042] In another example, the processor 120 may determine whether the driver request is a reinstall request from the loaded driver (block 720) to determine if the loaded driver is associated with a violation condition as shown in flow chart 700 of FIG. 7. That is, the loaded driver may request to uninstall and substitute a restricted protocol interface, which was previously installed to the processor system 100. Accordingly, the processor 120 may reject the reinstall request by the loaded driver (block 725).

[0043] In yet another example, the processor 120 may determine whether the driver request is an install request from the loaded driver (block 830) to determine if the loaded driver is associated with a violation condition as shown flow chart 800 in FIG. 8. The loaded driver may request to install a new protocol. If a restricted protocol interface is the same as the new protocol associated with the loaded driver (i.e., the restricted protocol

interface has the same GUID as the new protocol), the processor 120 may reject the install request by the loaded driver (block 825).

[0044] Alternatively, the processor 120 may implement a combination of blocks 510, 520, and 530. That is, the processor 120 may implement two of the blocks 510, 520, and 530. Referring to flow chart 900 of FIG. 9, for example, the processor 120 may determine whether the driver request is a reinstall request (block 920) and determine whether the driver request is an install request by the loaded driver (block 930) to determine if the loaded driver is associated with a violation condition. Accordingly, the processor 120 may reject the reinstall or install request by the loaded driver (block 925).

[0045] In another example, the processor 120 may determine whether the driver request is a request to access a restricted protocol interface (block 1010) and determine whether the driver request is a reinstall request (block 1020) to determine if the loaded driver is associated with a violation condition as shown in flow chart 1000 of FIG. 10. The processor 120 may store the restricted AP in the protocol database (block 1015) or reject the reinstall request by the loaded driver (block 1025).

[0046] As yet another example, the processor 120 may determine whether the driver request is a request to access a restricted protocol interface (block 1110) and determine whether the driver request is an install request by the loaded driver (block 1130) to determine if the loaded driver is associated with a violation condition as shown in flow chart 1100 of FIG. 11. The processor 120 may store the restricted protocol interface in the protocol database (block 1115) or reject the install request by the loaded driver (1125). As a result, the restricted protocol interfaces of the processor system 100 (i.e., the APs 320) are protected from unauthorized access (e.g., access by callers other than the phase core 330).

[0047] Although certain example methods, apparatus, and articles of manufacture have been described herein, the scope of coverage of this patent is not limited thereto. On the contrary, this patent covers all methods, apparatus, and articles of manufacture fairly falling within the scope of the appended claims either literally or under the doctrine of equivalents.